Full Inyection o Carburaciòn

Analisis de Algoritmos

Cristian Leon

Programa Ing. Ciencias de la Computación e Inteligencia Artificial
Universidad Sergio Arboleda

Bogotá, Colombia
Noviembre del 2025

Índice

1. Introducción

El presente informe desarrolla un modelo computacional orientado a la simulación del suministro de combustible en motores de combustión interna de pequeña cilindrada. El propósito principal es comparar el comportamiento de un sistema de inyección electrónica frente a un carburador, bajo un enfoque de análisis algorítmico. La simulación implementada utiliza técnicas de muestreo Monte Carlo y computación vectorizada, generando mapas de combustible en función de RPM y apertura del acelerador. Adicionalmente, se analiza la naturaleza NP-like del problema asociado a la generación exhaustiva de estos mapas, justificando el uso de métodos aproximados. El documento presenta la fundamentación teórica, el análisis del problema, el modelado, la arquitectura del software, la implementación y los resultados obtenidos.

Marco teórico

La operación eficiente de un motor de combustión interna depende críticamente de la correcta proporción entre masa de aire admitido y masa de combustible inyectada. La relación aire–combustible estequiométrica para gasolina convencional se aproxima por la razón:

$$\mathrm{AFR} = \frac{m_{\mathrm{aire}}}{m_{\mathrm{combustible}}} \approx 14{,}7$$

por cada masa de combustible (kg) se requieren $\approx 14{,}7$ kg de aire para la combustión completa. En electrónica de inyección la variable controlada es el *pulse width* (ancho de pulso) del inyector, que es proporcional a la masa de combustible necesaria y depende del caudal nominal del inyector. Matemáticamente:

$$\mathrm{pw}\ (\mathrm{s}) \propto \frac{m_{\mathrm{combustible}}}{\dot{m}_{\mathrm{inyector}}}.$$

Carburador: principios operativos El carburador es un dispositivo mecánico que dosifica combustible mediante diferencias de presión (efecto Venturi) y circuitos calibrados (chicler, aguja, circuito de baja/alta). Sus ventajas históricas son simplicidad, robustez y un comportamiento predecible en condiciones bien conocidas; además, cuando está correctamente afinado puede ofrecer mezclas adecuadas sin electrónica compleja. Por su naturaleza mecánica, el carburador responde directamente a la aspiración de aire y a la geometría de sus circuitos, y tiende a comportarse de manera estable en muchas condiciones reales.

Inyección electrónica en motocicletas: mito y realidad En las motocicletas de pequeña cilindrada, la inyección electrónica es típicamente del tipo inyección indirecta por puerto, por lo que el combustible no se inyecta dentro de la cámara de combustión —región donde ocurren los cuatro tiempos del ciclo Otto— sino en el conducto de admisión, aguas arriba de la válvula. Debido a esta arquitectura, el flujo de aire sigue dependiendo de un Venturi o estrangulamiento geométrico en el cuerpo de aceleración, similar al del carburador, que genera la depresión necesaria para modular la dinámica del aire aspirado y para asegurar una adecuada atomización del combustible. En consecuencia, aunque la dosificación del combustible está controlada electrónicamente mediante sensores (TPS, IAT, MAP/Baro, CKP) y una ECU, la base física del proceso —variación de presión inducida por el Venturi— sigue siendo comparable a la del carburador. Así, la ECU no "calcula" la mezcla desde primeros principios termodinámicos, sino que selecciona valores desde un mapa predefinido (RPM $\times$ TPS) con correcciones adicionales, lo cual limita la supuesta superioridad del sistema y explica por qué en motocicletas pequeñas la diferencia de rendimiento entre inyección y carburación puede ser menor de lo que suele asumirse.

Causas históricas de la migración a inyección La adopción masiva de la inyección no fue motivada exclusivamente por incrementos de potencia, sino por

factores regulatorios y de combustible: normas de emisiones más estrictas, necesidad de control preciso de mezcla para catalizadores, y la llegada de combustibles oxigenados (p. ej. con etanol) que favorecieron sistemas menos sensibles a depósitos y variantes químicas. El etanol y compuestos oxigenantes pueden producir residuos que afectan circuitos de carburador (pegajosidad, obstrucciones), mientras que los sistemas de inyección son menos vulnerables a esos efectos por su diseño (bombas y filtros cerrados, inyectores).

Mapas, muestreo y naturaleza NP-like Un *mapa de combustible* es una discretización del espacio de operación (p. ej. RPM $\times$ TPS) que asigna un ancho de pulso a cada celda. Si se añaden más variables o se incrementa la resolución, el número total de combinaciones crece exponencialmente (producto de niveles por sensor). Esa explosión combinatoria es la razón por la cual construir un mapa exhaustivo es computacionalmente impracticable; por ello se recurre a muestreo Monte Carlo y a técnicas de interpolación/optimización para aproximar la respuesta real del motor sin evaluar todas las combinaciones posibles. La distinción entre la velocidad de la implementación (ej., vectorización con NumPy) y la complejidad intrínseca del espacio de estados es central para este trabajo.

2. Análisis del problema

El objetivo central del proyecto es comparar el comportamiento de un sistema de carburación frente a un sistema de inyección electrónica en un motor monocilíndrico de motocicleta, mediante la construcción y análisis de mapas de combustible simulados. Para ello se modelan las variables físicas relevantes, se determina su rango de operación, se definen las discretizaciones necesarias y se establece la metodología de muestreo para aproximar el espacio total de estados del motor. La naturaleza combinatoria del problema requiere un enfoque probabilístico basado en Monte Carlo debido a la inviabilidad práctica de un muestreo exhaustivo.

Variables del sistema

En la simulación se consideran las variables fundamentales que determinan la masa de aire admitida y, por consiguiente, la masa de combustible requerida:

- **RPM (Revoluciones por minuto)**: indica la velocidad angular del cigüeñal y determina la frecuencia de ciclos de admisión.
- **TPS (Throttle Position Sensor)**: porcentaje de apertura del acelerador, que correlaciona con el flujo de aire admitido.
- **IAT (Intake Air Temperature)**: temperatura del aire de admisión, variable que influye en la densidad del aire según la relación inversa $\rho \propto \frac{1}{T}$.
- **Presión ambiental / MAP**: la presión absoluta afecta la densidad del aire y, por tanto, la masa admitida por ciclo.
- **VE (Volumetric Efficiency)**: eficiencia volumétrica del motor, modelada como una variable dependiente de RPM y carga. Representa qué fracción del volumen teórico del cilindro se llena realmente.

Estas variables constituyen un subconjunto esencial del conjunto completo de sensores que emplearía un sistema real de inyección; sin embargo, su combinación es suficiente para modelar la dinámica general del sistema y construir mapas representativos.

Rango y discretización

Cada variable se discretiza en un conjunto finito de niveles o *bins* que definen la resolución del mapa generado:

- **RPM**: típicamente entre 1500 y 9000 rpm, discretizadas en intervalos uniformes.

- **TPS**: entre 0 % y 100 %, dividido en niveles proporcionales (por ejemplo, 5–10 bins).
- **IAT**: temperatura entre 5°C y 35°C.
- **Presión**: valores típicos entre 70 kPa y 102 kPa, dependiendo de la altitud.
- **VE**: entre 65 % y 95 %, modelado como una variable continua dependiente de RPM.

El número total de combinaciones posibles resulta del producto cartesiano de todos los bins de todas las variables. Aun con discretizaciones conservadoras, este espacio crece exponencialmente, lo que refleja la naturaleza NP-like del problema. Por ejemplo, si se discretizan solo cinco variables con diez niveles cada una, se obtienen $10^5 = 100\,000$ combinaciones; aumentando la resolución a 20 niveles por variable se obtienen $3{,}2 \times 10^6$ combinaciones. Un mapa completo con más variables (p. ej. carga, temperatura de motor, barometría) puede alcanzar cientos de millones de estados.

Justificación del muestreo Monte Carlo

Dado que el espacio de estados crece de forma exponencial con la cantidad de variables y niveles de discretización, un muestreo exhaustivo resulta computacional y temporalmente inviable. Asimismo, muchos de los estados posibles representan situaciones operativas extremadamente improbables o físicamente irrelevantes para el funcionamiento real del motor.

Por estas razones, se implementa un **muestreo Monte Carlo**, donde cada estado se genera de manera aleatoria dentro del rango de cada variable, respetando distribuciones de probabilidad coherentes con el uso típico del motor. Este enfoque permite:

- Cubrir regiones representativas del espacio sin recorrerlo completamente.
- Reducir el costo computacional a tiempo lineal respecto al número de muestras.
- Conservar las correlaciones probabilísticas entre variables (por ejemplo, VE elevada a medio régimen, variaciones de IAT según entorno).
- Aproximar los mapas de combustible mediante densidad de muestras y promedios estadísticos.

En síntesis, el método Monte Carlo evita la explosión combinatoria y proporciona una aproximación robusta al comportamiento real del sistema con un costo computacional razonable.

Métricas evaluadas

Cada simulación produce varias métricas clave:

- **Pulso de inyección (ms)**: corresponde al ancho de pulso requerido para alcanzar la relación aire–combustible objetivo. Es la salida primaria del modelo.
- **Consumo en L/h**: a partir del pulso en milisegundos y del caudal nominal del inyector, se calcula la masa total de combustible consumida por hora.
- **Consumo por tanque**: convertido a litros y considerando un tanque de 11 L, se estima cuántas horas de operación continua puede sostener el motor.
- **Autonomía (km/tanque)**: usando una velocidad promedio representativa, se proyecta la distancia que podría recorrer la motocicleta antes de vaciar el tanque.

Estas métricas permiten una comparación directa entre carburación e inyección electrónica, no solo en términos de eficiencia de combustible, sino también en la estabilidad del sistema bajo variaciones ambientales y operativas.

3. Modelado del sistema y diagramas

El sistema propuesto requiere una caracterización clara de sus interacciones, funcionalidades y estructura interna. Para ello se incluyen cinco diagramas: contexto, casos de uso, actividades, NP-like y arquitectura. Cada uno aporta una perspectiva distinta que facilita comprender el funcionamiento del simulador de inyección y carburador.

3.1. Diagrama de contexto

El diagrama de contexto muestra el sistema como una caja negra, identificando al usuario como actor principal, las entradas que proporciona (parámetros y configuraciones de simulación) y las salidas que genera el sistema (mapas CSV, gráficos térmicos y resúmenes). Representa la interacción general sin detallar el funcionamiento interno.

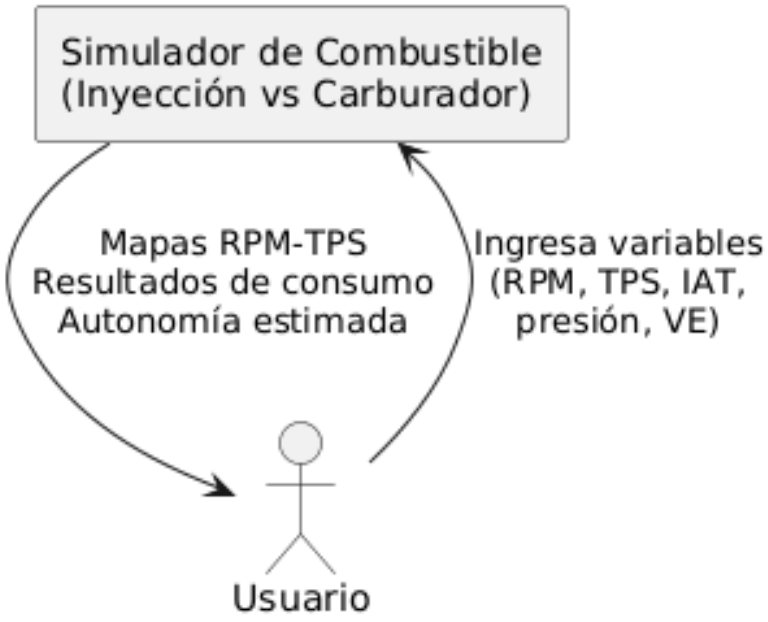


Figura 1: Diagrama de contexto.

3.2. Diagrama de casos de uso

Este diagrama resume las acciones que el usuario puede realizar dentro del sistema: seleccionar un piso térmico, definir la cantidad de muestras, ejecutar el motor de simulación, exportar resultados y visualizar el módulo NP-like. Permite entender las funciones principales disponibles para el operador.

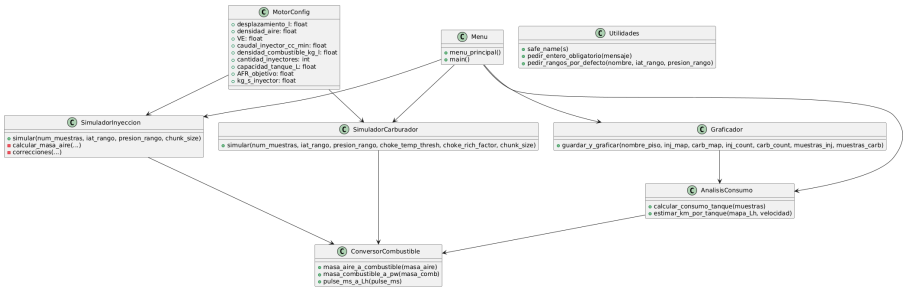


Figura 2: Casos de uso del sistema.

3.3. Diagrama de actividades (flujo de simulación)

Representa el flujo lógico desde que el usuario ingresa parámetros hasta que se generan los mapas. Incluye etapas como: generación de muestras, cálculo de masa de aire, obtención del pulse width, construcción del mapa térmico, conversión a L/h y exportación final de archivos.

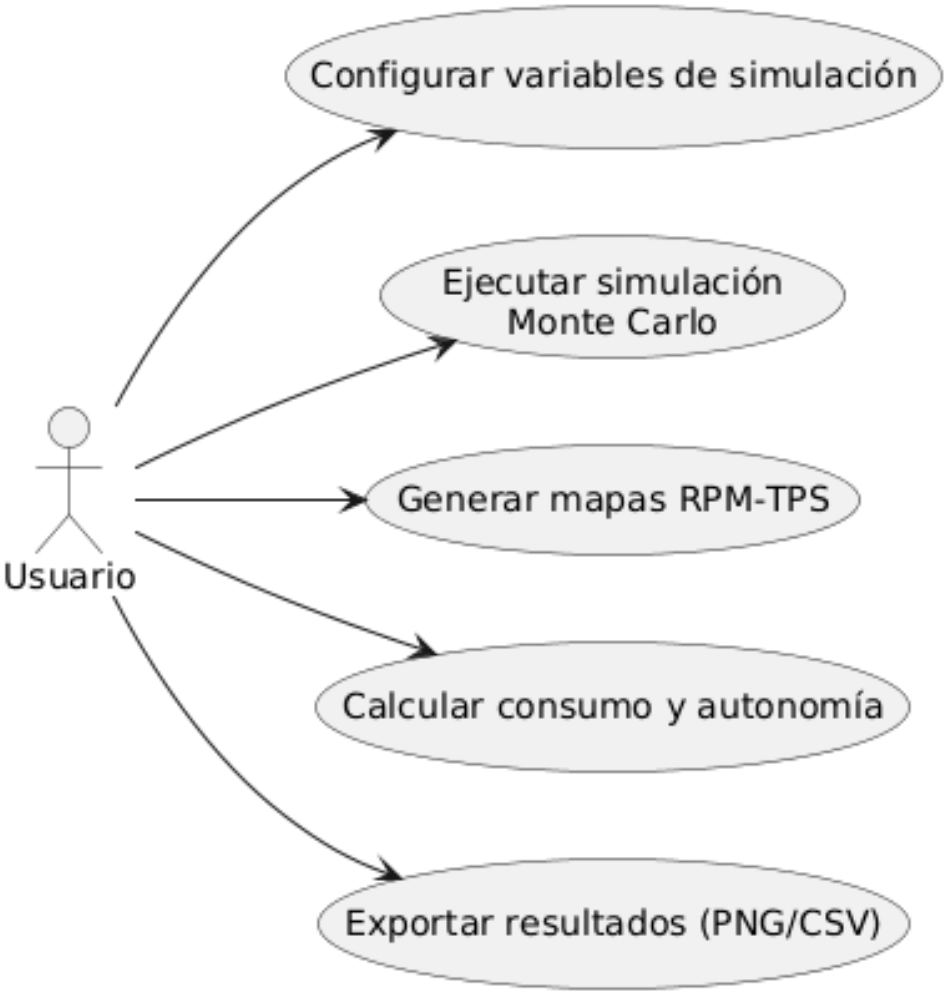


Figura 3: Flujo de actividades del proceso de simulación.

3.4. Diagrama de explosión combinatoria (NP-like)

Este diagrama ilustra cómo crece el número total de combinaciones a medida que aumenta la resolución de los sensores (RPM, TPS, IAT, presión). Permite visualizar el impacto computacional del modelo vectorizado y la necesidad de optimizar el procesamiento por lotes (chunking).

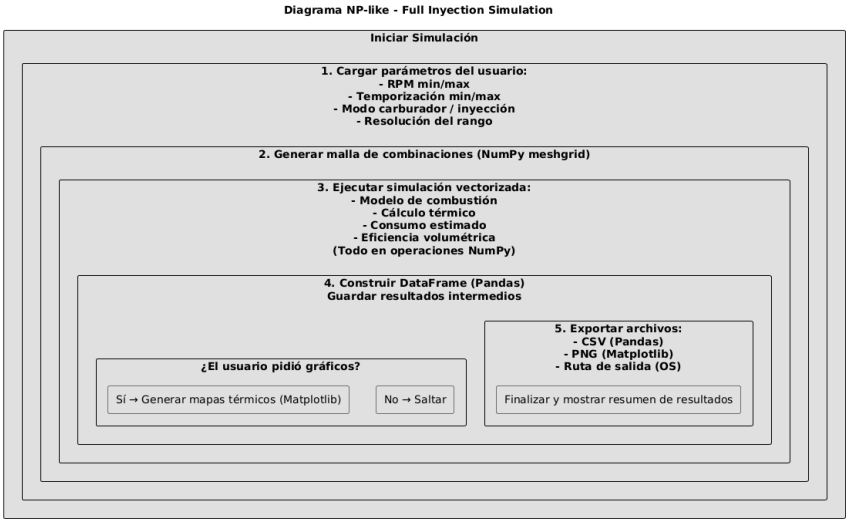


Figura 4: Diagrama NP-like que representa la explosión combinatoria.

3.5. Diagrama de arquitectura del sistema

Se presenta la estructura interna del simulador, organizada por módulos: interfaz, generador de muestras, simulador de inyección, simulador carburador, conversores de datos, exportación de archivos y módulo NP-like. Este diagrama ayuda a entender la organización del código y las dependencias entre componentes.

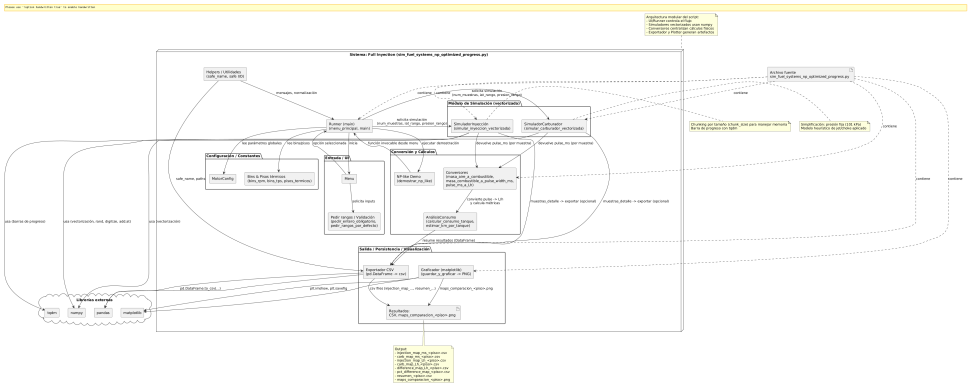


Figura 5: Arquitectura interna del sistema.

4. Diseño del software

El diseño del simulador se estructuró siguiendo un enfoque modular que permite aislar responsabilidades, facilitar el mantenimiento del código y habilitar futuras extensiones sin reescribir componentes centrales. Cada módulo cumple un rol específico dentro del proceso de simulación y se comunica con los demás mediante un flujo claro de datos. A continuación se describen los componentes principales del sistema.

4.1. Componentes del sistema

Módulo de entrada (UI / CLI) Este módulo gestiona la interacción inicial con el usuario. Permite definir el número de simulaciones para los sistemas de inyección y carburación, así como seleccionar las condiciones ambientales (modo normal o modo páramo). Su función principal es validar las entradas, establecer parámetros iniciales y dirigir el flujo hacia el motor de simulación.

Módulo de simulación vectorizada Constituye el núcleo matemático del sistema. Implementado con NumPy, genera muestras aleatorias para las variables principales (RPM, TPS, IAT, presión y eficiencia volumétrica). Utiliza operaciones vectorizadas para calcular, de manera simultánea, los valores de masa de aire y requerimiento de combustible bajo distintos regímenes de operación. Este enfoque evita recorrer exhaustivamente el espacio completo de estados, reduciendo la complejidad computacional y permitiendo simular miles de combinaciones en un solo paso.

Módulo de conversión y análisis A partir del *pulse width* calculado para cada muestra, este módulo convierte dicha magnitud a consumo volumétrico (L/h), aplicando el caudal nominal del inyector. También genera métricas comparativas como consumo promedio, autonomía estimada y dispersión de resultados. Incluye funciones para calcular estadísticas como percentiles, promedios y desviaciones, proporcionando una interpretación cuantitativa del comportamiento general.

Módulo de exportación Encargado de almacenar los resultados generados por los módulos anteriores. Produce archivos `CSV` con las métricas principales, así como gráficos `PNG` que incluyen mapas RPM–TPS, distribuciones y comparaciones visuales entre los sistemas. Su objetivo es permitir análisis posteriores sin necesidad de repetir las simulaciones.

Módulo opcional de calibración (GA) El diseño contempla la futura incorporación de un módulo basado en algoritmos genéticos (GA), utilizando librerías como DEAP. Este componente permitiría ajustar automáticamente parámetros del modelo con base en datos reales, buscando minimizar el error entre la simulación y

mediciones experimentales. Aunque no se implementa en la versión actual, la arquitectura habilita su integración sin modificaciones estructurales.

4.2. Integración general de los módulos

En conjunto, estos módulos conforman un flujo coherente que inicia con la adquisición de parámetros, continúa con el procesamiento matemático vectorizado, avanza hacia el análisis estadístico y finaliza con la exportación de resultados visuales y numéricos. Esta arquitectura modular permite una simulación robusta, escalable y fácilmente ampliable para futuras versiones.

5. Implementación

La implementación del simulador se desarrolló en Python utilizando un enfoque orientado al procesamiento vectorizado, con el fin de maximizar el rendimiento durante la generación masiva de muestras y el cálculo de parámetros de combustión. A continuación se describen las funciones principales incluidas en el sistema, sin mostrar aún el código fuente completo.

Función de generación de muestras Esta función crea arreglos vectorizados para las variables fundamentales del modelo: revoluciones por minuto (RPM), posición del acelerador (TPS), temperatura del aire (IAT), presión absoluta (MAP) y eficiencia volumétrica (VE). Emplea distribuciones uniformes o normales según la naturaleza de cada variable, produciendo miles de combinaciones aleatorias para alimentar el motor de simulación.

Función de cálculo de masa de aire Con base en las muestras generadas, esta función estima la masa de aire aspirada por el motor utilizando una versión simplificada de la ecuación de gas ideal. El cálculo se realiza completamente mediante operaciones vectorizadas de NumPy para reducir el tiempo de ejecución.

Función de cálculo del *pulse width* A partir de la masa de aire estimada, esta función determina el tiempo de apertura del inyector (*pulse width*) utilizando la relación estequiométrica y el caudal másico de combustible del sistema. Es uno de los pasos clave del simulador y está optimizado para operar sobre vectores de gran tamaño.

Función de conversión a consumo volumétrico Esta rutina convierte el *pulse width* calculado a unidades de consumo (L/h). Considera características del inyector, densidad del combustible y frecuencia de inyección por ciclo, generando valores comparables entre sistemas de inyección y carburación.

Función de análisis estadístico Incluye el cálculo de métricas derivadas tales como promedio, mediana, percentiles y desviación estándar. Estas estadísticas permiten caracterizar el comportamiento global del sistema bajo diferentes condiciones de operación y modos ambientales.

Función de exportación de resultados Encargada de generar archivos externos en formatos `CSV` y gráficos `PNG`. Produce mapas RPM–TPS, distribuciones de consumo y comparaciones entre los sistemas modelados. Facilita la documentación y análisis de los resultados sin necesidad de ejecutar nuevamente la simulación.

Funciones auxiliares El código incluye funciones adicionales para verificación de parámetros, normalización de datos, control del modo páramo y manejo interno de rutas para la exportación de archivos. Aunque no forman parte del núcleo matemático, garantizan consistencia y robustez en la ejecución.

```
# archivo: sim_fuel_systems_np_optimized_progress.py
# Requisitos: numpy, pandas, matplotlib, tqdm
# pip install numpy pandas matplotlib tqdm

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from tqdm import tqdm
import time
import os
import math
import unicodedata

# ---------------- Par metros generales ----------------
SEED = 42
np.random.seed(SEED)

# Motor ajustado para moto 125cc
desplazamiento_l = 0.125      # litros (125cc)
densidad_aire = 1.2           # kg/m3 aprox
VE = 0.9                      # volumetric efficiency (0-1)
caudal_inyector_cc_min = 300.0  # cc/min por inyector (ejemplo)
densidad_combustible_kg_l = 0.75  # kg/L (gasolina)
cantidad_inyectores = 1       # inyectores por motor
capacidad_tanque_L = 11.0     # litros

# Mapeo base (RPM x TPS)
bins_rpm = np.linspace(1000, 12000, 25)    # 25 bins -> 24 celdas
bins_tps = np.linspace(0.0, 100.0, 25)     # 25 bins -> 24 celdas

# Rangos por defecto para pisos t rmicos (sin tierra helada)
pisos_termicos = {
    1: ("Tierra caliente", (24, 38), (95, 101)),
    2: ("Tierra templada", (17, 24), (90, 101)),
    3: ("Tierra fr a", (12, 17), (85, 101)),
    4: ("P ramo", (6, 12), (70, 95)),
}

# Conversi n caudal inyector cc/min -> kg/s
def inyector_ccmin_a_kg_s(cc_min, densidad=densidad_combustible_kg_l):
    L_min = cc_min / 1000.0
    kg_min = L_min * densidad
    return kg_min / 60.0

kg_s_inyector = inyector_ccmin_a_kg_s(caudal_inyector_cc_min) *
    cantidad_inyectores

AFR_objetivo = 14.7  # relaci n aire/combustible objetivo

```

```
# ---------------- Helpers ----------------
def safe_name(s):
    """Convierte nombre a min sculas, sin tildes ni espacios (para archivos)."""
    s_nf = unicodedata.normalize('NFKD', s)
    s_ascii = ''.join(c for c in s_nf if not unicodedata.combining(c))
    return s_ascii.replace(' ', '_').lower()

def pedir_rangos_por_defecto(nombre_piso, iat_rango, presion_rango):
    print(f"\nHas seleccionado '{nombre_piso}'.")
    print("Los rangos recomendados para este piso t rmico son:")
    print(f"  Temperatura de admisi n (IAT): {iat_rango[0]} C  a {iat_rango[1]} C ")
    print(f"  Presi n ambiente: {presion_rango[0]} kPa a {presion_rango[1]} kPa")
    input("Presiona Enter para aceptar estos rangos y continuar...")
    return iat_rango, presion_rango

# Entrada obligatoria (sin default)
def pedir_entero_obligatorio(mensaje):
    while True:
        entrada = input(mensaje + ": ").strip()
        if entrada == "":
            print("Debes ingresar un n mero entero positivo (no se aceptan valores por defecto).")
            continue
        try:
            val = int(entrada)
            if val > 0:
                return val
            else:
                print("Por favor ingresa un n mero entero positivo.")
        except:
            print("Entrada inv lida. Intenta de nuevo.")

# ---------------- Conversiones ----------------
def masa_aire_a_combustible(masa_aire_kg_s, afr=AFR_objetivo):
    return masa_aire_kg_s / afr

def masa_combustible_a_pulse_width_ms(masa_comb_kg_s, kg_s_inyector_local=kg_s_inyector):
    if np.isscalar(masa_comb_kg_s):
        if kg_s_inyector_local <= 0:
            return 0.0
    else:
        if kg_s_inyector_local <= 0:
            return np.zeros_like(masa_comb_kg_s)
    pw_s = masa_comb_kg_s / kg_s_inyector_local
    return pw_s * 1000.0

def pulse_ms_a_Lh(pulse_ms, kg_s_inyector_local=kg_s_inyector, densidad=densidad_combustible_kg_l):
    duty = pulse_ms / 1000.0  # seg
```

```
    flujo_kg_s = kg_s_inyector_local * duty
    flujo_L_s = np.divide(flujo_kg_s, densidad, out=np.zeros_like(flujo_kg_s), where=(densidad!=0))
    return flujo_L_s * 3600.0  # L/h

# ---------------- Simulaciones vectorizadas (NP-like, mapa 2D) ----------------
def simular_inyeccion_vectorizada(num_muestras, iat_rango, presion_rango, chunk_size=200000):
    """
    Simulaci n vectorizada con barra de progreso por chunks.
    """
    shape = (len(bins_rpm)-1, len(bins_tps)-1)
    suma_map = np.zeros(shape, dtype=float)
    cuenta_map = np.zeros(shape, dtype=int)
    muestras_list = []

    total_chunks = math.ceil(num_muestras / chunk_size)
    processed = 0

    for _ in tqdm(range(total_chunks), desc="Simulaci n INYECCI N", unit="chunk"):
        n = min(chunk_size, num_muestras - processed)
        if n <= 0:
            break

        rpms = np.random.uniform(1000.0, 12000.0, size=n)
        tps = np.random.uniform(0.0, 100.0, size=n)
        iat = np.random.uniform(iat_rango[0], iat_rango[1], size=n)
        presion = np.random.uniform(presion_rango[0], presion_rango[1], size=n)

        ciclos_por_seg = rpms / 2.0 / 60.0
        vol_por_ciclo_m3 = (desplazamiento_l / 1000.0) * VE
        flujo_vol_m3_s = vol_por_ciclo_m3 * ciclos_por_seg * (tps / 100.0)
        masa_flujo_kg_s = flujo_vol_m3_s * densidad_aire

        corr_temp = 1.0 - 0.01 * (iat - 20.0)
        corr_presion = presion / 101.0
        masa_aire_corr = masa_flujo_kg_s * corr_temp * corr_presion

        masa_comb_req = masa_aire_corr / AFR_objetivo  # kg/s

        with np.errstate(divide='ignore', invalid='ignore'):
            pw_s = masa_comb_req / kg_s_inyector
        pulse_ms = np.where(kg_s_inyector > 0, pw_s * 1000.0, 0.0)

        idx_rpm = np.digitize(rpms, bins_rpm) - 1
        idx_tps = np.digitize(tps, bins_tps) - 1
        idx_rpm = np.clip(idx_rpm, 0, shape[0]-1)
        idx_tps = np.clip(idx_tps, 0, shape[1]-1)

```

```
        pos_plana = idx_rpm * shape[1] + idx_tps
        np.add.at(suma_map.ravel(), pos_plana, pulse_ms)
        np.add.at(cuenta_map.ravel(), pos_plana, 1)

        muestras_chunk = np.column_stack((rpms, tps, iat, presion, pulse_ms))
        muestras_list.append(muestras_chunk)

        processed += n

    with np.errstate(divide='ignore', invalid='ignore'):
        mapa_promedio = np.divide(suma_map, np.where(cuenta_map>0, cuenta_map, 1))
    mapa_promedio[cuenta_map==0] = np.nan
    muestras_detalle = np.vstack(muestras_list) if len(muestras_list) > 0 else np.empty((0,5))
    return mapa_promedio, cuenta_map, muestras_detalle

def simular_carburador_vectorizada(num_muestras, iat_rango, presion_rango, choke_temp_thresh=5.0, choke_rich_factor=1.5, chunk_size=200000):
    shape = (len(bins_rpm)-1, len(bins_tps)-1)
    suma_map = np.zeros(shape, dtype=float)
    cuenta_map = np.zeros(shape, dtype=int)
    muestras_list = []

    total_chunks = math.ceil(num_muestras / chunk_size)
    processed = 0

    for _ in tqdm(range(total_chunks), desc="Simulaci n CARBURADOR", unit="chunk"):
        n = min(chunk_size, num_muestras - processed)
        if n <= 0:
            break

        rpms = np.random.uniform(1000.0, 8000.0, size=n)
        tps = np.random.uniform(0.0, 100.0, size=n)
        iat = np.random.uniform(iat_rango[0], iat_rango[1], size=n)
        presion = np.full(n, 101.0)  # presi n fija para carburador (simplificado)

        ciclos_por_seg = rpms / 2.0 / 60.0
        vol_por_ciclo_m3 = (desplazamiento_l / 1000.0) * VE
        flujo_vol_m3_s = vol_por_ciclo_m3 * ciclos_por_seg * (tps / 100.0)
        masa_flujo_kg_s = flujo_vol_m3_s * densidad_aire

        jet_base_flow = 0.5
        jet_effect = jet_base_flow * (1.0 + 0.01 * tps) * (1.0 + 0.0001 * (rpms - 2000.0))
        masa_comb_req = masa_flujo_kg_s / AFR_objetivo * jet_effect

        choke_mask = (iat <= choke_temp_thresh)
```

```
187         if choke_rich_factor != 1.0:
188             masa_comb_req = masa_comb_req * np.where(choke_mask,
    choke_rich_factor, 1.0)
189 
190         with np.errstate(divide='ignore', invalid='ignore'):
191             pw_s = masa_comb_req / kg_s_inyector
192         pulse_ms = np.where(kg_s_inyector > 0, pw_s * 1000.0, 0.0)
193 
194         idx_rpm = np.digitize(rpms, bins_rpm) - 1
195         idx_tps = np.digitize(tps, bins_tps) - 1
196         idx_rpm = np.clip(idx_rpm, 0, shape[0]-1)
197         idx_tps = np.clip(idx_tps, 0, shape[1]-1)
198 
199         pos_plana = idx_rpm * shape[1] + idx_tps
200         np.add.at(suma_map.ravel(), pos_plana, pulse_ms)
201         np.add.at(cuenta_map.ravel(), pos_plana, 1)
202 
203         muestras_chunk = np.column_stack((rpms, tps, iat, presion,
    pulse_ms))
204         muestras_list.append(muestras_chunk)
205 
206         processed += n
207 
208     with np.errstate(divide='ignore', invalid='ignore'):
209         mapa_promedio = np.divide(suma_map, np.where(cuenta_map>0,
    cuenta_map, 1))
210     mapa_promedio[cuenta_map==0] = np.nan
211     muestras_detalle = np.vstack(muestras_list) if len(muestras_list) >
     0 else np.empty((0,5))
212     return mapa_promedio, cuenta_map, muestras_detalle
213 
214 # ---------------- Consumo y kilometraje ----------------
215 def calcular_consumo_tanque(muestras_detalle, duracion_seg=3600):
216     if muestras_detalle is None or len(muestras_detalle) == 0:
217         return 0.0, 0.0
218     pulse_ms = muestras_detalle[:, 4]  # ms por muestra
219     Lh_por_muestra = pulse_ms_a_Lh(pulse_ms, kg_s_inyector)
220     consumo_medio_Lh = np.nanmean(Lh_por_muestra)
221     if np.isnan(consumo_medio_Lh) or consumo_medio_Lh <= 0:
222         return 0.0, 0.0
223     horas_por_tanque = capacidad_tanque_L / consumo_medio_Lh
224     km_por_tanque = horas_por_tanque * 40.0  # velocidad promedio
    estimada 40 km/h
225     return consumo_medio_Lh, km_por_tanque
226 
227 def estimar_km_por_tanque(mapa_Lh, velocidad_promedio_kmh=40):
228     if mapa_Lh is None or np.isnan(mapa_Lh).all():
229         return 0.0
230     consumo_medio_Lh = np.nanmean(mapa_Lh)
231     if consumo_medio_Lh <= 0 or np.isnan(consumo_medio_Lh):
232         return 0.0
233     horas_por_tanque = capacidad_tanque_L / consumo_medio_Lh
234     return horas_por_tanque * velocidad_promedio_kmh
```

```
235
236 # ---------------- NP-like: demostrar explosi n combinatoria
    ----------------
237 def demostrar_np_like():
238     print("\n--- Demostraci n de crecimiento NP-like ---")
239     print("Al aumentar la resoluci n del mapa o el n mero de sensores
    ,")
240     print("el n mero de combinaciones posibles crece exponencialmente
    .\n")
241     print("1) N mero de celdas en el mapa (RPM    TPS):")
242     for bins in [10, 20, 25, 30, 40]:
243         celdas = (bins - 1) * (bins - 1)
244         print(f"  {bins} bins -> {celdas} celdas (aprox.)")
245     print("\n2) Combinaciones con S sensores y L niveles:")
246     sensores = [2, 3, 4, 5]
247     niveles = [10, 20, 50, 100]
248     for s in sensores:
249         print(f"  Con {s} sensores:")
250         for l in niveles:
251             combinaciones = l ** s
252             print(f"    {l}^{s} = {combinaciones:.2e} combinaciones")
253     print("\n3) Tiempo de muestreo ilustrativo:")
254     print("  Si cada muestra toma 1 ms, y queremos cubrir 1e6
    combinaciones:")
255     print("  Tiempo = 1e6 ms = 1000 s = ~16.7 minutos")
256     print("  Para 1e9 combinaciones = ~11.5 d as")
257
258 # ---------------- Guardado y gr ficos ----------------
259 def guardar_y_graficar(nombre_piso, inj_map_ms, carb_map_ms, inj_count,
     carb_count, muestras_inj, muestras_carb, outdir="resultados"):
260     nombre_safe = safe_name(nombre_piso)
261     os.makedirs(outdir, exist_ok=True)
262
263     inj_Lh = pulse_ms_a_Lh(inj_map_ms, kg_s_inyector)
264     carb_Lh = pulse_ms_a_Lh(carb_map_ms, kg_s_inyector)
265
266     diff_Lh = inj_Lh - carb_Lh
267     with np.errstate(divide='ignore', invalid='ignore'):
268         pct_diff = 100.0 * diff_Lh / np.where(np.abs(carb_Lh) > 1e-12,
    carb_Lh, np.nan)
269
270     # Guardar mapas
271     pd.DataFrame(inj_map_ms).to_csv(os.path.join(outdir, f"
    injection_map_ms_{nombre_safe}.csv"), index=False, header=False)
272     pd.DataFrame(carb_map_ms).to_csv(os.path.join(outdir, f"
    carb_map_ms_{nombre_safe}.csv"), index=False, header=False)
273     pd.DataFrame(inj_Lh).to_csv(os.path.join(outdir, f"
    injection_map_Lh_{nombre_safe}.csv"), index=False, header=False)
274     pd.DataFrame(carb_Lh).to_csv(os.path.join(outdir, f"carb_map_Lh_{
    nombre_safe}.csv"), index=False, header=False)
275     pd.DataFrame(diff_Lh).to_csv(os.path.join(outdir, f"
    difference_map_Lh_{nombre_safe}.csv"), index=False, header=False)
```

```
    pd.DataFrame(pct_diff).to_csv(os.path.join(outdir, f"pct_difference_map_{nombre_safe}.csv"), index=False, header=False)

    # Resumen CSV por piso (una sola fila)
    consumo_inj_Lh, km_inj = calcular_consumo_tanque(muestras_inj)
    consumo_carb_Lh, km_carb = calcular_consumo_tanque(muestras_carb)
    celdas_inj = int(np.nansum(~np.isnan(inj_map_ms)))
    celdas_carb = int(np.nansum(~np.isnan(carb_map_ms)))

    resumen = {
        "nombre": [nombre_safe],
        "consumo_iny_Lh": [consumo_inj_Lh],
        "km_iny": [km_inj],
        "consumo_carb_Lh": [consumo_carb_Lh],
        "km_carb": [km_carb],
        "celdas_iny": [celdas_inj],
        "celdas_carb": [celdas_carb]
    }
    resumen_df = pd.DataFrame(resumen)
    resumen_path = os.path.join(outdir, f"resumen_{nombre_safe}.csv")
    resumen_df.to_csv(resumen_path, index=False)

    print(f"\nGuardados resultados en '{outdir}' para {nombre_piso}")
    print(f"  Consumo inyecci n (estimado): {consumo_inj_Lh:.4f} L/h ({km_inj:.2f} km/tanque)")
    print(f"  Consumo carburador (estimado): {consumo_carb_Lh:.4f} L/h ({km_carb:.2f} km/tanque)")

    # Graficar mapas (L/h)
    plt.figure(figsize=(15,10))
    plt.subplot(2,3,1)
    plt.title(f"Mapa Inyecci n (ms) - {nombre_piso}")
    plt.imshow(inj_map_ms, origin='lower', aspect='auto', cmap='viridis')
    plt.colorbar()
    plt.subplot(2,3,2)
    plt.title(f"Mapa Carburador (ms) - {nombre_piso}")
    plt.imshow(carb_map_ms, origin='lower', aspect='auto', cmap='plasma')
    plt.colorbar()
    plt.subplot(2,3,3)
    plt.title(f"Mapa Inyecci n (L/h) - {nombre_piso}")
    plt.imshow(inj_Lh, origin='lower', aspect='auto', cmap='viridis')
    plt.colorbar()
    plt.subplot(2,3,4)
    plt.title(f"Mapa Carburador (L/h) - {nombre_piso}")
    plt.imshow(carb_Lh, origin='lower', aspect='auto', cmap='plasma')
    plt.colorbar()
    plt.subplot(2,3,5)
    plt.title(f"Diferencia absoluta (L/h) - {nombre_piso}")
    plt.imshow(diff_Lh, origin='lower', aspect='auto', cmap='seismic')
    plt.colorbar()
    plt.subplot(2,3,6)
```

```
    plt.title(f"Diferencia relativa (%) - {nombre_piso}")
    plt.imshow(pct_diff, origin='lower', aspect='auto', cmap='bwr')
    plt.colorbar()

    plt.tight_layout()
    fig_path = os.path.join(outdir, f"maps_comparacion_{nombre_safe}.
   png")
    plt.savefig(fig_path)
    plt.close()
    print(f"  Imagen generada: {fig_path}")
    print(f"  Resumen guardado: {resumen_path}")

# ---------------- Men  e inicio ----------------
def menu_principal():
    print("Simulaci n de sistemas de combustible para moto 125cc (NP-
   like, vectorizado)")
    print("Seleccione un piso t rmico:")
    for k in sorted(pisos_termicos.keys()):
        print(f"  {k}) {pisos_termicos[k][0]}")
    print("  5) Todas")
    print("  6) Demostrar NP-like")
    print("  0) Salir")

    opcion = input("Ingrese opci n: ").strip()
    if opcion == "0":
        print("Saliendo...")
        exit(0)
    elif opcion == "5":
        return "todas", None, None
    elif opcion == "6":
        return "np", None, None
    else:
        try:
            op_int = int(opcion)
            if op_int in pisos_termicos:
                return "uno", op_int, None
            else:
                print("Opci n inv lida.")
                return menu_principal()
        except:
            print("Opci n inv lida.")
            return menu_principal()

def main():
    opcion, piso_seleccionado, _ = menu_principal()

    if opcion == "np":
        demostrar_np_like()
        return

    if opcion == "todas":
        num_inyeccion = pedir_entero_obligatorio("N mero de
   simulaciones para INYECCI N (entero positivo)")
```

```
374         num_carburador = pedir_entero_obligatorio("N mero de
    simulaciones para CARBURADOR (entero positivo)")
375
376         for k in pisos_termicos:
377             nombre_piso, iat_rango, presion_rango = pisos_termicos[k]
378             print(f"\nEjecutando simulaci n para piso t rmico: {
    nombre_piso}")
379
380             inj_map_ms, inj_count, muestras_inj =
    simular_inyeccion_vectorizada(num_inyeccion, iat_rango,
    presion_rango)
381             carb_map_ms, carb_count, muestras_carb =
    simular_carburador_vectorizada(num_carburador, iat_rango,
    presion_rango)
382
383             guardar_y_graficar(nombre_piso, inj_map_ms, carb_map_ms,
    inj_count, carb_count, muestras_inj, muestras_carb)
384
385     elif opcion == "uno":
386         nombre_piso, iat_rango, presion_rango = pisos_termicos[
    piso_seleccionado]
387         iat_rango, presion_rango = pedir_rangos_por_defecto(nombre_piso
    , iat_rango, presion_rango)
388
389         num_inyeccion = pedir_entero_obligatorio("N mero de
    simulaciones para INYECCI N (entero positivo)")
390         num_carburador = pedir_entero_obligatorio("N mero de
    simulaciones para CARBURADOR (entero positivo)")
391
392         print("\nIniciando simulaci n de INYECCI N (vectorizada)...")
393         inj_map_ms, inj_count, muestras_inj =
    simular_inyeccion_vectorizada(num_inyeccion, iat_rango,
    presion_rango)
394         print("Inyecci n completada.")
395
396         print("\nIniciando simulaci n de CARBURADOR (vectorizada)...")
397         carb_map_ms, carb_count, muestras_carb =
    simular_carburador_vectorizada(num_carburador, iat_rango,
    presion_rango)
398         print("Carburador completado.")
399
400         guardar_y_graficar(nombre_piso, inj_map_ms, carb_map_ms,
    inj_count, carb_count, muestras_inj, muestras_carb)
401
402 if __name__ == "__main__":
403     main()
```

Listing 1: Simulación vectorizada (extracto)

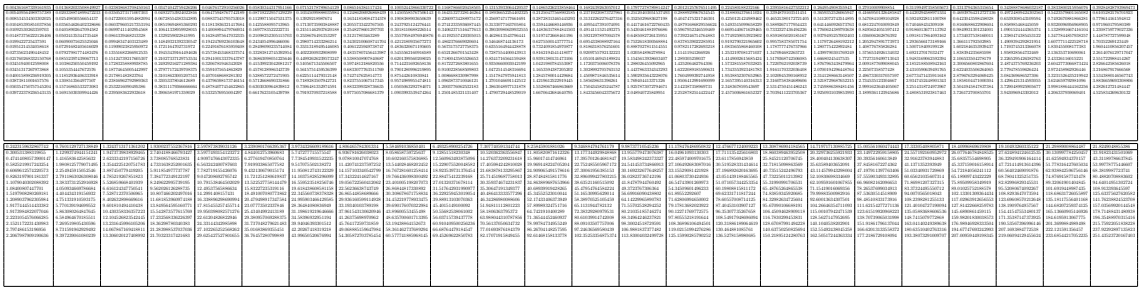
% ———————— RESULTADOS ————————-

6. Resultados

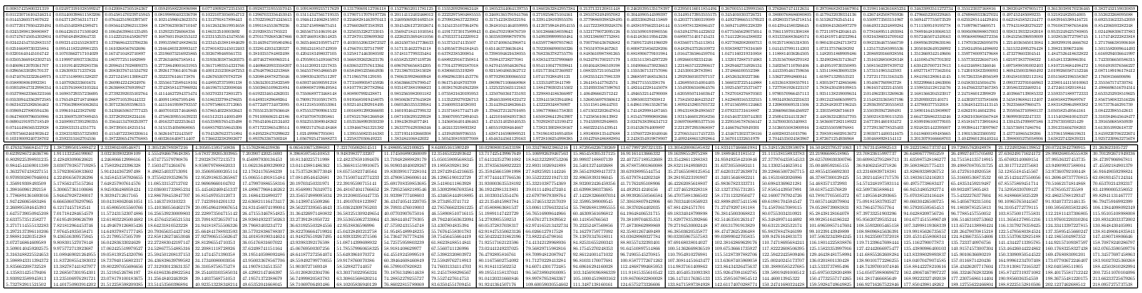
En esta sección se presentan los resultados obtenidos para cada uno de los ecosistemas evaluados: páramo, tierra caliente, tierra fría y tierra templada. Se incluyen mapas de carburación e inyección, diferencias absolutas y porcentuales, tablas resumen y mapas comparativos en formato PNG.

6.1. Resultados en Páramo

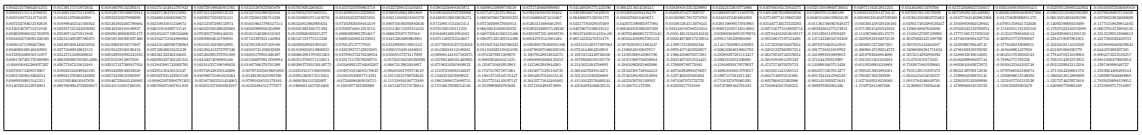
Mapas de carburación



Mapas de inyección



Diferencias absolutas



Porcentajes de diferencia

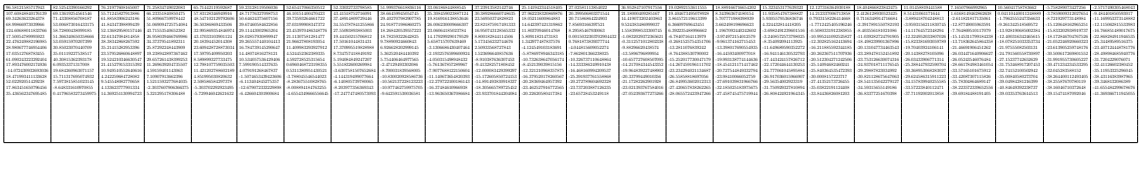


Tabla resumen

nombre	consumo$_i nyL h$	km$_i ny$	consumo$_c arb_L h$	km$_c arb$	celdas$_i ny$	celdas$_c arb$
paramo	1.0982164338656561	400.64962281726777	0.94178605567334	467.1974036453718	576	384

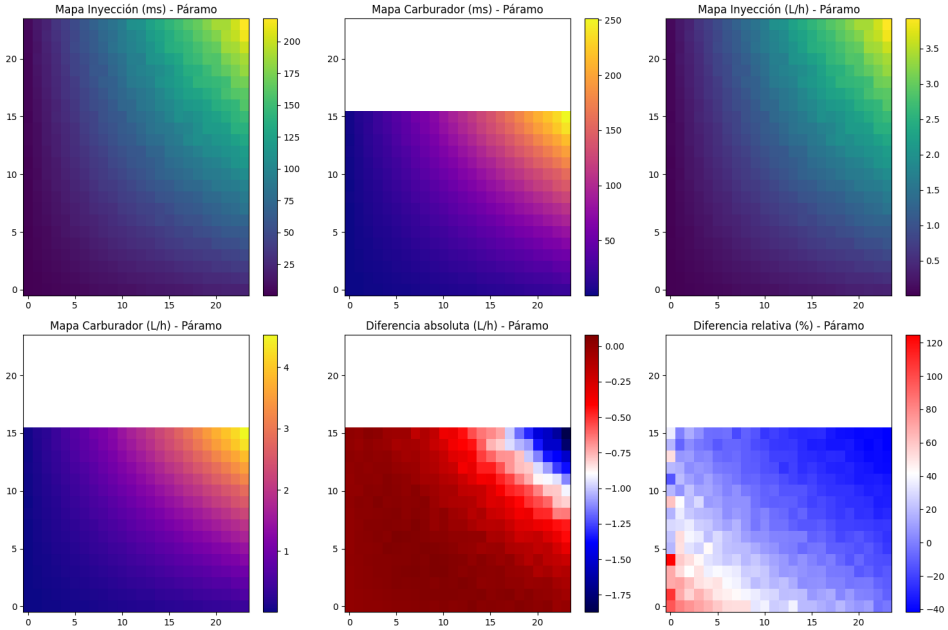
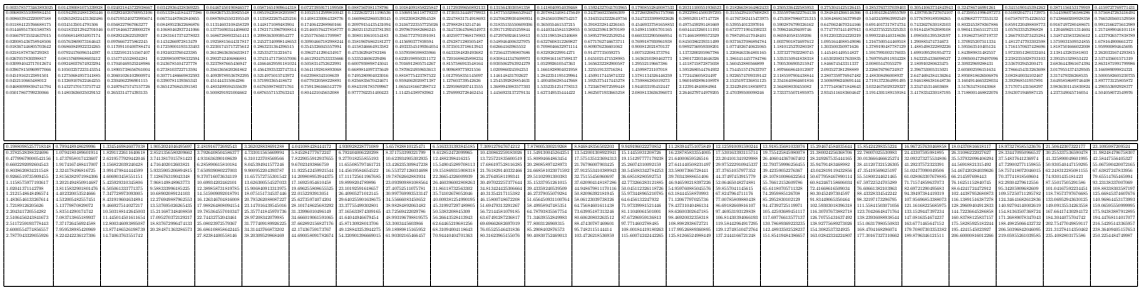


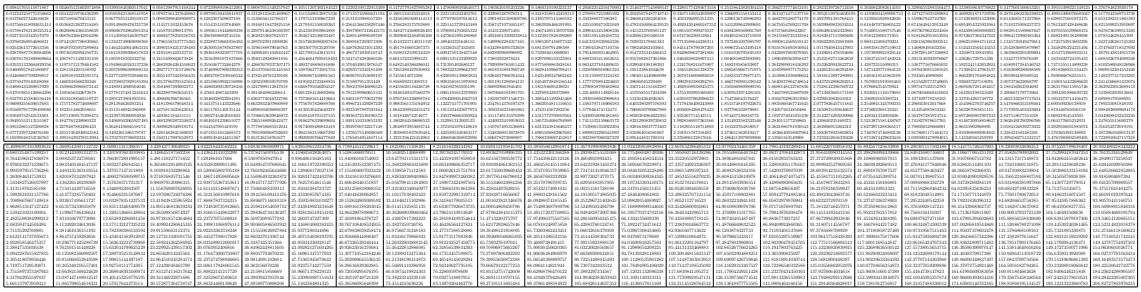
Figura 6: Mapa comparativo entre mapas de inyección y carburación en ecosistema de páramo.

6.2. Resultados en Tierra Caliente

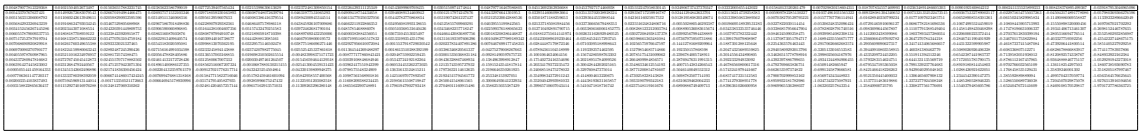
Mapas de carburación



Mapas de inyección



Diferencias absolutas



Porcentajes de diferencia

Tabla resumen

nombre	consumo$_i nyl h$	km$_i ny$	consumo$_c arb l h$	km$_c arb$	celdas$_i ny$	celdas$_c arb$
tierra$_c aliente$	1.029093137617957	427.56091155992783	0.9226021942841363	476.91193748070805	576	383

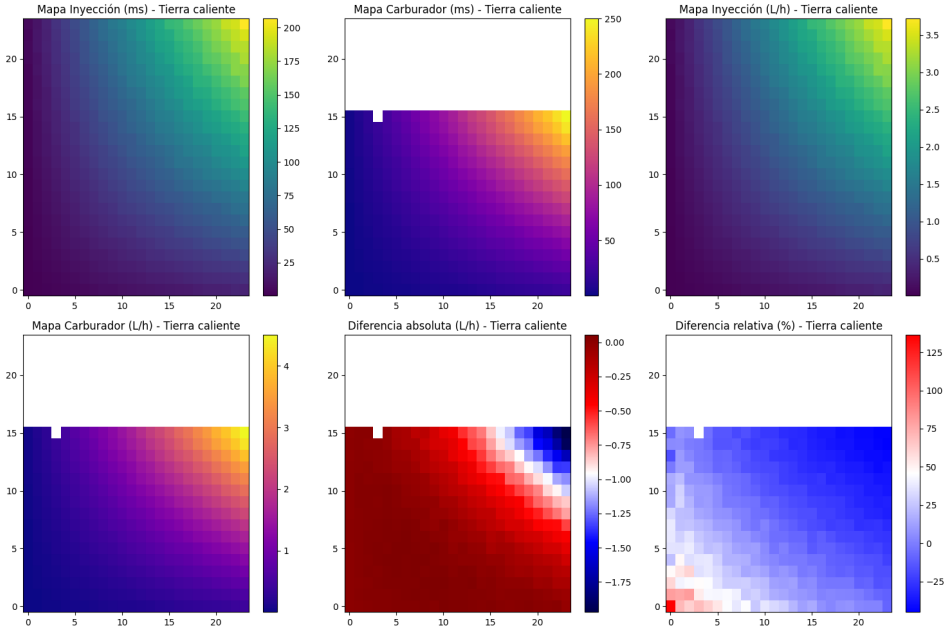
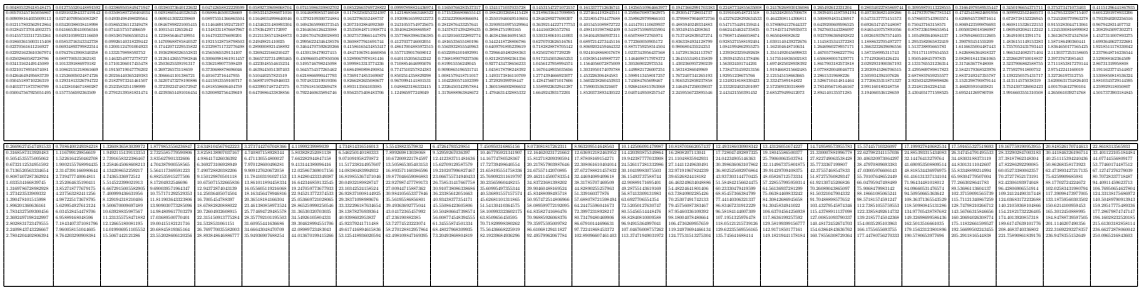


Figura 7: Mapa comparativo entre inyección y carburación en tierra caliente.

6.3. Resultados en Tierra Fría

Mapas de carburación



Mapas de inyección

Diferencias absolutas

Porcentajes de diferencia

Tabla resumen

nombre	consumo$_i nyL h$	km$_i ny$	consumo$_c arbL h$	km$_c arb$	celdas$_i ny$	celdas$_c arb$
tierra$_f ria$	1.1765594691460832	373.97174689294775	0.9371383759848774	469.5144402101624	576	384

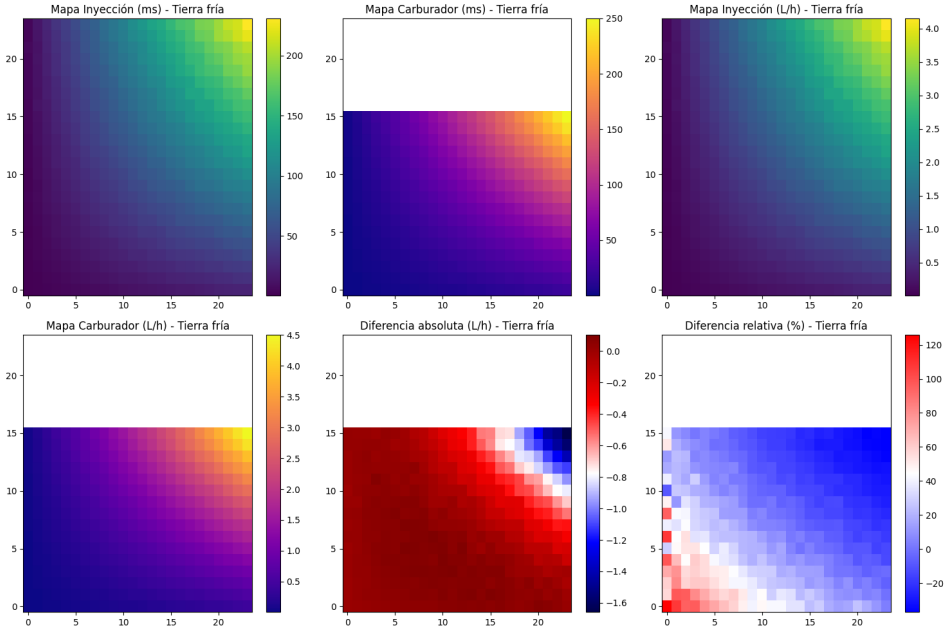
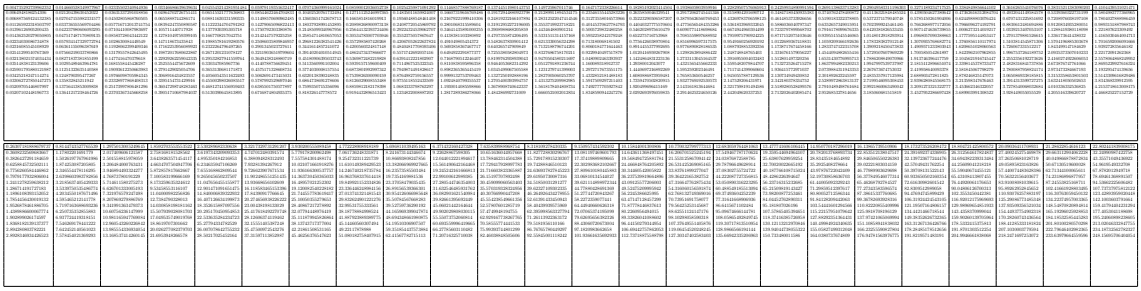


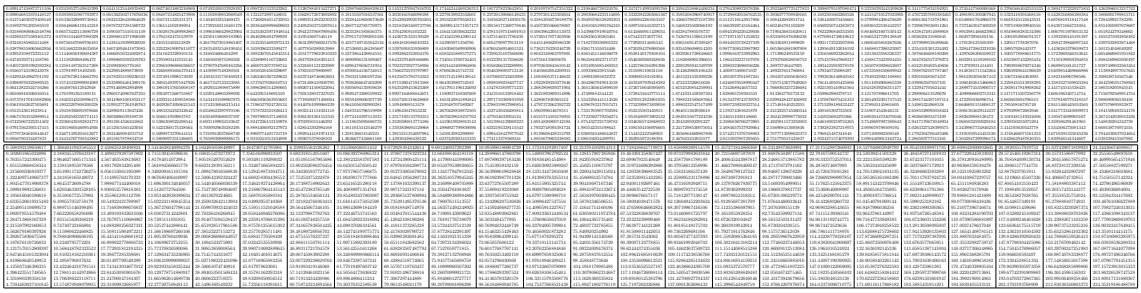
Figura 8: Mapa comparativo entre inyección y carburación en tierra fría.

6.4. Resultados en Tierra Templada

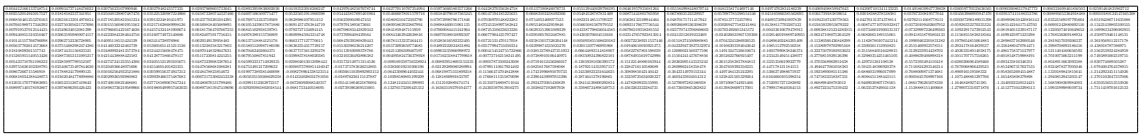
Mapas de carburación



Mapas de inyección



Diferencias absolutas



Porcentajes de diferencia

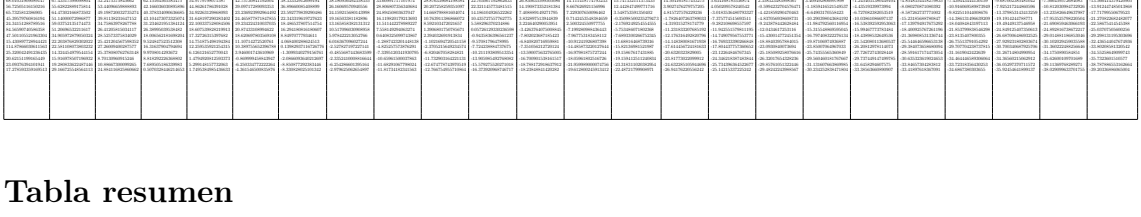


Tabla resumen

nombre	consumo$_i nyLh$	km$_i ny$	consumo$_c arbL h$	km$_c arb$	celdas$_i ny$	celdas$_c arb$
tierra$_t emplada$	1.1227918295472765	391.88030089016	0.9130536611158437	481.8993874492279	576	384

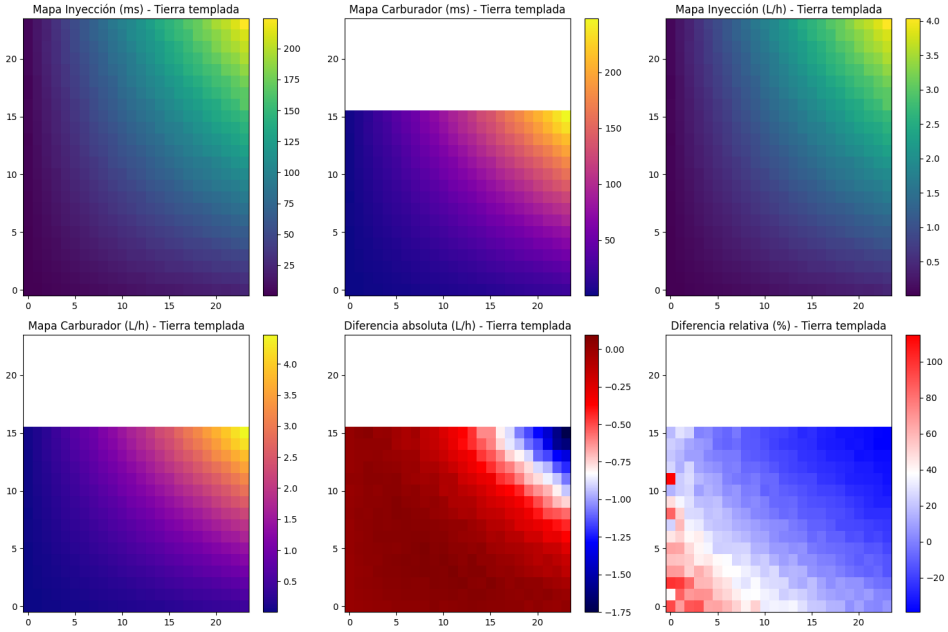


Figura 9: Mapa comparativo entre inyección y carburación en tierra templada.

7. Conclusiones

Las figuras y mapas generados muestran de forma coherente que el sistema de inyección tiende a producir respuestas más suaves y corregidas frente a las variaciones de RPM, apertura de mariposa y condiciones ambiente, mientras que el carburador exhibe patrones más toscos y picos de riqueza, especialmente en condiciones de baja temperatura donde el efecto del choke aumenta notablemente el consumo. Al convertir los tiempos de apertura equivalentes (pulse width en ms) a litros por hora, queda claro que en muchas celdas operativas el carburador entrega mayor flujo de combustible que la inyección, lo que se traduce en una menor autonomía por tanque; en las condiciones simuladas la inyección presenta un consumo medio menor y, por tanto, mayor kilometraje estimado para un tanque de 11 L. Estas diferencias quedan acentuadas en pisos térmicos fríos y de alta altitud, donde la inyección compensa variaciones de densidad del aire mediante correcciones y el carburador, con una geometría fija y la intervención del choke, tiende a sobrerreponer mezcla. Es importante resaltar que los resultados provienen de un modelo simplificado y de muestreos Monte Carlo con supuestos heurísticos (p. ej. factor de choke, rango de muestreo distinto para carburador hasta 8000 rpm), por lo que los valores absolutos son estimativos y las zonas vacías en algunos mapas son artefactos de muestreo. Desde la perspectiva NP-like, la calibración fina de mapas es intrínsecamente combinatoria: discretizar cada celda y permitir múltiples niveles por celda genera un espacio de soluciones que crece exponencialmente, de ahí la necesidad de métodos heurísticos; la integración de un algoritmo genético en la simulación demuestra una vía práctica para buscar soluciones aceptables en subespacios manejables sin recurrir a búsquedas exhaustivas. En términos prácticos, los mapas en ms y en L/h deberían acompañarse en el informe con la estimación de autonomía por tanque y con una discusión sobre sensibilidad por regiones del mapa, priorizando la calibración en las celdas que más contribuyen al consumo y validando finalmente con datos reales de logs para afinar el modelo y las estrategias de optimización.

Bibliografía

Referencias

[1] Hondamaquina, "Carburación o inyección: cómo funciona cada sistema de suministro de combustible", Hondamaquina.com. Disponible en:

https://hondamaquina.com/carburacion-o-inyeccion-como-funciona-cada-sis
(Consulta: Noviembre del 2025).

[2] Motor.es, "¿Qué es la inyección?", Motor.es. Disponible en:

https://www.motor.es/que-es/inyeccion.

(Consulta: Noviembre del 2025).

[3] YouTube, "Carburador vs. inyector de combustible - Por qué los motociclistas deberían pensarlo 2 veces". Disponible en:

https://www.youtube.com/watch?v=cYI8iEHoh-A&t=304s.

(Consulta: Noviembre del 2025).